



CREDIT ASSIGNMENT FOR ENERGY-AWARE MULTI-AGENT REINFORCEMENT LEARNING: A SHAPLEY-VALUE APPROACH

Madhav

A report submitted as part of the requirements of the Summer Internship
at the Department of Computer Science & Engineering

Motilal Nehru National Institute of Technology

Prayagraj, India

July 2026

Supervisor: Dr. Shailendra Shukla

Certificate



This is to certify that the report titled “Credit Assignment for Energy-Aware Multi-Agent Reinforcement Learning: A Shapley-Value Approach” submitted by **Madhav** has been prepared under my guidance and supervision as part of the Summer Internship programme. The experiments, code, and results described herein were carried out by the student over the course of the internship.

Dr. Shailendra Shukla

Project Supervisor

Department of Computer Science & Engineering

MNNIT Prayagraj

Abstract

Cooperative multi-agent reinforcement learning (MARL) tasks such as coverage and area-sweep problems require dividing a single team-level reward among individual agents so that each agent receives a training signal proportional to its actual contribution. This project studies the credit-assignment problem in a modified Simple-Spread coverage environment in which agents additionally carry a finite energy budget that depletes with movement. Four credit-assignment schemes are implemented and compared under identical Deep Q-Network (DQN) training conditions: (i) uniform splitting of the team reward, (ii) single-counterfactual difference rewards, (iii) Monte Carlo-estimated Shapley values over a pure coverage value function ($\lambda = 0$), and (iv) an energy-aware Shapley value function that additionally penalises the coalition’s mean energy expenditure ($\lambda = 0.5$).

Each method is trained for 500 episodes across three random seeds. Learning curves, Bellman-loss convergence, Jain’s fairness index over residual agent energy, the distribution of raw difference-reward values, and qualitative rollout trajectories are used to characterise the behaviour of each scheme. The results show that both Shapley-based methods converge to a materially better team reward than the difference-reward baseline, that difference rewards suffer from a credit-starvation failure mode in which only the nearest agent to a landmark receives a non-trivial signal, and that introducing an explicit energy term into the coalition value function changes the fairness profile of the learned policies, in some rollouts producing agents that conserve energy by remaining near their starting position once nearby landmarks are already covered by teammates. These findings motivate a discussion of how coalition value functions shape emergent multi-agent behaviour beyond raw task reward.

Acknowledgments

I would like to express my sincere gratitude to my project supervisor, **Dr. Shailendra Shukla**, for their guidance, feedback, and encouragement throughout this internship. Their willingness to engage deeply with the technical details of the credit-assignment experiments, question unclear results, and suggest additional diagnostics substantially improved both the rigour of the work and my own understanding of multi-agent reinforcement learning.

I am also thankful to the faculty and staff of the Department of Computer Science & Engineering at MNNIT for providing the computational resources, laboratory access, and overall academic environment that made this work possible. The department's support in making compute time available was particularly valuable given the number of training runs required across the four credit-assignment schemes and three random seeds evaluated in this study.

I would further like to thank my fellow interns and peers, whose discussions during the course of this project often helped clarify implementation details of the Shapley-value estimation procedure and the energy-aware environment dynamics. Their feedback during informal presentations of intermediate results was instrumental in shaping the final structure of the experiments and this report.

Finally, I thank my family for their patience, encouragement, and unwavering support throughout the duration of this internship. Their constant motivation gave me the confidence and perseverance to work through the more challenging aspects of this project, from debugging training instabilities to interpreting the fairness metrics reported in the later chapters of this report.

Madhav

Declaration

I, **Madhav**, hereby declare that the project report entitled “Credit Assignment for Energy-Aware Multi-Agent Reinforcement Learning: A Shapley-Value Approach” submitted by me is a record of my own original work, carried out under the guidance of **Dr. Shailendra Shukla**, and has not been submitted previously for any degree or diploma at any other institution.

I further declare that all the experiments, implementations, and analyses described in this report, including the modified Simple-Spread coverage environment, the four credit-assignment schemes, and the associated DQN training pipeline, were designed and carried out by me during the course of the Summer Internship. Where existing libraries, frameworks, or published methods have been used or adapted, they have been explicitly acknowledged and cited within the body of this report.

I understand that any information used from other sources has been duly cited and referenced, and that this report does not, to the best of my knowledge, infringe upon the intellectual property rights of any third party. I take full responsibility for the accuracy and authenticity of the results, figures, and conclusions presented herein.

This report is submitted in partial fulfilment of the requirements of the Summer Internship programme and represents my own effort and contribution, carried out under the supervision and guidance of **Dr. Shailendra Shukla**, except where explicit references have been made to the work of others.

Signed:

Date:

Contents

1	Introduction	1
2	Profile of the Problem and Rationale / Scope of the Study	3
2.1	Profile of the Problem	3
2.2	Rationale and Scope of the Study	4
3	Problem Analysis	5
3.1	Problem Definition	5
4	System Design	7
4.1	Overview	7
4.2	System Architecture	7
4.3	Module Descriptions	7
4.4	Technology Stack	8
4.4.1	Deployment / Execution Design	8
5	Software Requirement Analysis	9
5.1	Introduction	9
5.2	Functional Requirements	9
5.2.1	Environment Configuration	9
5.2.2	Credit-Assignment Mode Selection	9
5.2.3	Reproducibility	9
5.2.4	Batch Execution	10
5.2.5	Result Aggregation	10

5.3	Non-Functional Requirements	10
5.3.1	Performance	10
5.3.2	Modularity	10
5.3.3	Maintainability	10
5.4	Summary	10
6	Implementation	11
6.1	Energy-Augmented Environment (energy_env.py)	11
6.2	Coalition Value Function and Credit-Assignment Engine (shapley_engine.py)	12
6.3	DQN Model (models.py)	13
6.4	Experiment Runner (run_all.py)	14
6.5	Summary	14
7	Experimental Setup	15
8	Results	16
8.1	Learning Curves	16
8.2	DQN Loss Convergence	17
8.3	Fairness of Residual Energy	17
8.4	Distribution of Raw Difference-Reward Values	18
8.5	Environment Rollouts under Energy-Aware Shapley	18
8.6	Summary Table	22
9	Executive Summary	23
9.1	Comparison of Methods	23
9.2	Reconciling Fairness and Rollout Behaviour	23
9.3	Difference-Reward Underperformance	24
10	Conclusions	25
10.1	Key Findings	25
10.1.1	Recommendations	25
10.2	Future Work	26

10.3 Final Verdict	26
A Full Source Listings	29
A.1 energy_env.py	29
A.2 shapley_engine.py	31
A.3 models.py	33
A.4 run_all.py	34
B Extended Discussion	36
B.1 Naive and Difference-Based Rewards	36
B.2 Shapley-Value-Based Credit Assignment	37
B.3 Energy- and Resource-Aware Extensions	37
B.4 Implications for This Project’s Findings	37
C Threats to Validity and Limitations	39
C.1 Sample Size	39
C.2 Rollout Sample Size	39
C.3 Fixed Hyperparameters	40
C.4 Environment Scope	40

List of Tables

7.1	Experiment configuration parameters shared across all twelve training runs.	15
8.1	Final-episode summary statistics across four credit-assignment methods (mean $\pm$ std over 3 seeds).	22
9.1	Qualitative comparison of the four credit-assignment methods.	23
9.2	Per-seed rollout observations for Energy-Aware Shapley ($\lambda = 0.5$).	24

List of Figures

8.1	Learning Curves (mean $\pm$ std over 3 seeds) for different reward-splitting mechanisms.	16
8.2	DQN Training Loss Convergence showing Avg. MSE Bellman Loss across 3 seeds.	17
8.3	Jain’s fairness index profile monitored across 500 episodes for evaluating inequality in energy consumption.	18
8.4	Histogram of raw values (D_i) demonstrating severe credit starvation near zero.	19
8.5	Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 1.	19
8.6	Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 2.	20
8.7	Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 3.	21

Chapter 1

Introduction

Cooperative multi-agent reinforcement learning problems are typically formulated with a single, shared team reward that captures how well the group as a whole is performing a task. When agents are trained independently — for example, each running its own Deep Q-Network — this team reward must somehow be turned into a per-agent training signal. The naive approach of simply giving every agent the full team reward, or an equal share of it, ignores the fact that agents can contribute very differently to the outcome in any given episode: one agent may be doing all of the useful work while another is idle or actively unhelpful, yet under uniform splitting both receive identical credit.

This project investigates the credit-assignment problem in a coverage-style multi-agent environment built on PettingZoo’s Simple-Spread task (mpe2), extended with a battery/energy model so that each agent’s ability to act is a finite, depletable resource. Four agents (or five, depending on configuration) must cover a set of landmarks while managing their own energy budgets, since an agent that exhausts its battery is forced to remain stationary for the rest of the episode. This setting is used to compare four ways of converting the shared team reward into individual learning signals: uniform splitting, single-counterfactual difference rewards, Monte Carlo Shapley values computed over a pure coverage value function, and an energy-aware Shapley value function that additionally accounts for how much energy a coalition of agents has consumed.

The central research question is whether principled game-theoretic credit assignment (Shapley values) produces measurably better training outcomes than cheaper heuristics (uniform splitting, difference rewards), and how introducing an explicit energy term into the underlying coalition value function changes not just the reward signal but the emergent behaviour and fairness properties of the trained policies. All four methods are trained under identical DQN hyperparameters and evaluated using the same suite of metrics — team reward, Bellman loss convergence, Jain’s fairness index over residual energy, the distribution of raw difference-reward values, and qualitative rollout visualisation — so

that differences in outcome can be attributed to the credit-assignment mechanism rather than to confounding factors in the training pipeline.

The remainder of this report is organised as follows. Chapter 2 profiles the problem and motivates the scope of the study. Chapter 3 gives a formal problem analysis. Chapter 4 describes the system design, including the energy-augmented environment and the Shapley credit-assignment engine. Chapter 5 covers software requirements. Chapter 6 walks through the implementation in detail, referencing the actual training and engine code. Chapter 7 documents the experimental setup. Chapter 8 presents the results: learning curves, loss convergence, fairness metrics, the difference-reward histogram, and rollout visualisations. Chapter 9 discusses these results, including an executive summary comparing the four methods. Chapter 10 concludes the report and outlines directions for future work.

Chapter 2

Profile of the Problem and Rationale / Scope of the Study

2.1 Profile of the Problem

In multi-agent coverage tasks, the standard reward signal is a single scalar that describes the quality of the joint configuration of all agents relative to a set of landmarks (for example, the negative sum of minimum agent-to-landmark distances). Training independent per-agent value functions from this single scalar requires a credit-assignment rule. Poorly chosen rules can create two related failure modes:

- (a) agents that receive too little differentiated signal learn slowly or converge to a degenerate policy such as “follow the average behaviour,” and
- (b) agents that are structurally advantaged by the value function (for example, whichever agent happens to already be closest to a landmark) receive a disproportionate share of the credit regardless of what the other agents did, discouraging exploration by the remaining agents.

The addition of a finite energy budget compounds this problem: a self-interested credit-assignment rule may inadvertently reward an agent for conserving energy by doing nothing, if the value function it is judged against does not clearly separate “the coalition performed well because I helped” from “the coalition performed well and I happened to save my battery.” This project’s problem profile is therefore two-fold: first, evaluate whether Shapley-based credit assignment outperforms cheaper alternatives on the base coverage task; second, evaluate how an energy penalty embedded in the coalition value function reshapes the fairness and idleness properties of the resulting policies.

2.2 Rationale and Scope of the Study

- **Knowledge Retrieval:** Characterise how each credit-assignment method distributes signal across agents by directly inspecting the coalition value function and the resulting per-agent credit shares.
- **Comparative Evaluation:** Train all four methods under identical DQN hyperparameters, network architecture, and environment configuration, varying only the reward decomposition rule.
- **Fairness Analysis:** Quantify how evenly residual energy is distributed across agents at the end of training using Jain’s fairness index, to detect free-riding or idle-agent behaviour.
- **Failure-Mode Diagnosis:** Use the raw difference-reward distribution to test the theoretical prediction that single-counterfactual difference rewards under-credit all but the nearest agent.
- **Qualitative Validation:** Inspect rollout trajectories under the energy-aware policy across multiple seeds to check whether patterns seen in the aggregate metrics are reproducible or seed-specific artefacts.

The scope of the study is limited to a fixed environment size (5 agents, 3–5 landmarks depending on seed), a fixed training budget of 500 episodes per run, and three random seeds per method (12 runs total). It does not attempt to tune the energy-penalty weight λ beyond the two values used (0 and 0.5), nor does it explore alternative RL algorithms beyond DQN; these are identified as future work in Chapter 10.

Chapter 3

Problem Analysis

3.1 Problem Definition

Let N be the set of agents and $S \subseteq N$ a coalition. A coalition value function $v(S)$ assigns a scalar utility to any subset of agents being “present” in the environment. In this project:

$$v(S) = \text{coverage}(S) - \lambda \cdot \text{mean\_energy\_used}(S)$$

where $\text{coverage}(S)$ is the negative sum, over all landmarks, of the minimum distance from any agent in S to that landmark, and $\text{mean\_energy\_used}(S)$ is the average fraction of battery already consumed by the agents in S . Setting $\lambda = 0$ recovers a pure coverage value function; $\lambda = 0.5$ additionally penalises coalitions whose members have depleted more of their energy.

Given $v(\cdot)$, the four credit-assignment rules studied are:

1. **Uniform Splitting:** every agent receives an equal share of the team reward at each step, irrespective of $v(\cdot)$.
2. **Difference (Counterfactual) Reward:** $D_i = v(N) - v(N \setminus \{i\})$, i.e. the drop in value if agent i were removed from the grand coalition; only one counterfactual coalition is evaluated per agent.
3. **Monte Carlo Shapley ($\lambda = 0$):** the Shapley value ϕ_i is estimated by averaging agent i ’s marginal contribution $v(S \cup \{i\}) - v(S)$ over K random agent orderings, using the pure coverage value function.
4. **Energy-Aware Shapley ($\lambda = 0.5$):** identical Monte Carlo Shapley estimation procedure, but computed over the energy-penalised value function.

Ideal / Reality / Consequences / Proposal (IRCP) Analysis

Ideal: An ideal credit-assignment rule would give every agent a signal that is monotonic in, and proportional to, its true marginal contribution to team performance, would be fair with respect to energy usage so that no agent is structurally incentivised to free-ride, and would be computationally cheap enough to evaluate at every environment step during training.

Reality: No single method studied here satisfies all three properties simultaneously. Uniform splitting is cheap but not proportional. Difference rewards are cheap relative to full Shapley estimation but, as shown in Chapter 8, systematically under-credit agents other than the nearest one to a landmark. Monte Carlo Shapley is proportional in expectation but requires K sampled orderings per step, and its energy-aware variant can create an incentive to idle once nearby landmarks are already covered by teammates.

Consequences: Difference rewards plateau at a substantially worse team reward than the two Shapley variants (Chapter 8, Figure 8.1) and exhibit a large mass of near-zero credit values (Figure 8.4), consistent with the credit-starvation prediction above. The energy-aware value function achieves competitive team reward but does not straightforwardly improve energy fairness relative to the pure-coverage baselines (Table 8.2), and rollout inspection shows at least one agent per episode in two of the three seeds electing to remain near its starting position rather than travel to a landmark (Chapter 8, Figures 8.5–8.7).

Proposal: Sections 9.1 and 10.1 propose that a variance-based (rather than purely mean-based) energy penalty in $v(S)$ could more directly discourage idle free-riding, since the current mean formulation cannot distinguish “the coalition used little energy because everyone contributed efficiently” from “the coalition used little energy because one member did nothing.”

Chapter 4

System Design

4.1 Overview

The system consists of three main components: (1) an energy-augmented wrapper around PettingZoo’s Simple-Spread environment, (2) a coalition value function and Monte Carlo Shapley / difference-reward credit-assignment engine, and (3) a set of independent per-agent DQN learners trained using whichever credit signal the active method produces. A top-level experiment runner sweeps over all method $\times$ seed combinations sequentially.

4.2 System Architecture

At each environment step, the `EnergyWrapper` advances the underlying `mpe2` Simple-Spread environment, deducts energy from each acting agent, and appends each agent’s residual energy fraction to its own observation vector. Depleted agents are automatically overridden to the “stay” action for the remainder of the episode. In parallel, the `PettingZooShapleyEngine` reads agent positions and landmark positions directly from the underlying MPE world state to evaluate the coalition value function $v(S)$ for arbitrary subsets S , which it uses to compute either Monte Carlo Shapley or single-counterfactual difference rewards, depending on the active mode.

4.3 Module Descriptions

- `energy_env.py` — `EnergyWrapper`: battery bookkeeping, observation augmentation, depleted-agent action override.
- `shapley_engine.py` — `PettingZooShapleyEngine`: coalition value function $v(S)$, Monte Carlo Shapley estimation, single-counterfactual difference rewards, credit

normalisation.

- `models.py` — DQN: a 2-hidden-layer fully connected Q-network and an epsilon-greedy action-selection helper.
- `train.py` — orchestrates environment reset/step, credit computation, DQN updates, and CSV logging for a single (mode, seed) combination.
- `run_all.py` — sweeps the full method $\times$ seed experiment matrix sequentially and reports per-run completion status.

4.4 Technology Stack

- Python 3 with PyTorch for the DQN implementation.
- PettingZoo / mpe2 (`simple_spread_v3`) as the base multi-agent coverage environment.
- NumPy for coalition-value computation and vectorised distance calculations.
- Matplotlib for learning-curve, fairness-index, histogram, and rollout visualisation.

4.4.1 Deployment / Execution Design

Let M be the number of methods (4) and S_r be the number of random seeds (3). The sequential grid execution space defines a structured computational pipeline running in $O(M \cdot S_r)$ sequential blocks.

Experiments are designed to run identically on any OS: `run_all.py` invokes `train.py` as a subprocess once per (method, seed) pair using `sys.executable`, so the same script works unmodified on Windows, Linux, or macOS. An optional tmux-based parallel runner (`run_all.sh`) is provided for Unix-like systems where the 12 runs can be distributed across panes instead of executing sequentially.

Chapter 5

Software Requirement Analysis

5.1 Introduction

This chapter documents the functional and non-functional requirements that guided the design of the experiment pipeline, so that the four credit-assignment methods could be compared under strictly controlled and reproducible conditions.

5.2 Functional Requirements

5.2.1 Environment Configuration

The system must support instantiating the Simple-Spread environment with a configurable number of agents, landmarks, and maximum episode length, and must augment it with a configurable energy model (starting energy, per-step movement cost, per-step stay cost).

5.2.2 Credit-Assignment Mode Selection

The training entry point must accept a mode argument selecting among uniform, difference, shapley, and shapley_energy, and must route the appropriate value function ($\lambda = 0$ or $\lambda = 0.5$) to the Shapley engine accordingly.

5.2.3 Reproducibility

Every run must accept an explicit random seed argument and log results (per-episode reward, loss, fairness index, energy state) to a dedicated CSV file so that runs can be re-analysed without re-training.

5.2.4 Batch Execution

The system must be able to launch the complete method $\times$ seed matrix ($4 \times 3 = 12$ runs) from a single command, with a smoke-test mode that runs a small number of episodes to verify the pipeline end-to-end before committing to full 500-episode runs.

5.2.5 Result Aggregation

Per-run CSV logs must be aggregable into a single summary table (mean $\pm$ std across seeds, per method) and into the figures described in Chapter 8.

5.3 Non-Functional Requirements

5.3.1 Performance

Coalition-value evaluation is the dominant per-step cost for the Shapley-based methods, since Monte Carlo estimation requires K sampled orderings, each needing up to $|N|$ incremental value-function evaluations. The number of samples (`MC_SHAPLEY_SAMPLES`) is therefore a configurable trade-off between estimator variance and wall-clock training time.

5.3.2 Modularity

The energy wrapper, the Shapley engine, and the DQN model are implemented as independent modules with narrow interfaces, so that, for example, the value function or the network architecture can be modified without touching the other components.

5.3.3 Maintainability

All per-agent residual energy bookkeeping is centralised in `EnergyWrapper` so that `train.py`, `shapley_engine.py`, and any downstream analysis scripts read energy state from a single source of truth rather than re-implementing battery logic independently.

5.4 Summary

The requirements above ensured that differences observed in Chapter 8 can be attributed to the credit-assignment mechanism itself, rather than to inconsistencies in environment configuration, random seeding, or logging across the four methods.

Chapter 6

Implementation

This chapter walks through the implementation of the three core modules — the energy wrapper, the Shapley credit-assignment engine, and the DQN model — and the top-level experiment runner.

6.1 Energy-Augmented Environment (`energy_env.py`)

`EnergyWrapper` wraps PettingZoo’s `simple_spread_v3.parallel_env` with `local_ratio=0.0`. Each agent begins an episode with `energy_start` units of energy. On every step, a movement action costs `move_cost` and the designated “stay” action costs `stay_cost`; once an agent’s energy reaches zero it is marked depleted and any action it selects is silently overridden to “stay” for the rest of the episode. The wrapper also augments every agent’s observation vector with its own current energy fraction, so that the Q-network can condition its policy on remaining battery.

The core implementation logic for state tracking and step updates is given in Listing 6.1.

```
1 def energy_fraction(self, agent):
2     return max(0.0, self.energy[agent]) / self.energy_start
3
4 def step(self, actions):
5     effective_actions = {}
6     for agent, action in actions.items():
7         if self.is_depleted(agent):
8             effective_actions[agent] = self.stay_action_index
9         else:
10            effective_actions[agent] = action
11    next_obs, rewards, terminations, truncations, infos = self.
        env.step(effective_actions)
12    for agent, action in effective_actions.items():
```

```

13         if self.energy[agent] <= 0.0:
14             continue
15         cost = self.stay_cost if action == self.stay_action_index
16             else self.move_cost
17         self.energy[agent] = max(0.0, self.energy[agent] - cost)
18     return self._augment(next_obs), rewards, terminations,
19         truncations, infos

```

Listing 6.1: Core step logic of EnergyWrapper — depleted-agent override and energy deduction.

6.2 Coalition Value Function and Credit-Assignment Engine (shapley_engine.py)

PettingZooShapleyEngine implements $v(S) = \text{coverage}(S) - \lambda \cdot \text{mean\_energy\_used}(S)$ directly against the underlying MPE world state: `coverage(S)` sums, over every landmark, the minimum Euclidean distance from any agent in S to that landmark (negated, so that closer coverage yields higher value); `mean_energy_used(S)` is the mean, over agents in S , of one minus their residual energy fraction as reported by the `EnergyWrapper`.

```

1 def _estimate_v(self, coalition, env, num_landmarks):
2     if len(coalition) == 0:
3         return 0.0
4     agent_positions = [mpe_env.world.agents[i].state.p_pos for i
5         in coalition]
6     landmark_positions = [lm.state.p_pos for lm in mpe_env.world.
7         landmarks]
8     total_proximity = 0.0
9     for lm_pos in landmark_positions:
10         min_dist_to_lm = min([np.linalg.norm(a_pos - lm_pos) for
11             a_pos in agent_positions])
12         total_proximity += min_dist_to_lm
13     coverage = -float(total_proximity)
14     if self.lam == 0.0 or self.energy_wrapper is None:
15         return coverage
16     energy_used = [1.0 - self.energy_wrapper.energy_fraction(name
17         ) for name in agent_names]
18     mean_energy_used = float(np.mean(energy_used))
19     return coverage - self.lam * mean_energy_used

```

Listing 6.2: Coalition value function $v(S)$, shared by both Shapley estimation and difference rewards.

Monte Carlo Shapley estimation draws $K = \text{num\_samples}$ random agent orderings; for each ordering it accumulates every agent’s marginal contribution $v(S \cup \{i\}) - v(S)$ as it is inserted, then averages over all K samples. The difference-reward rule instead evaluates only two coalitions per agent — the grand coalition N and N minus that agent — which is far cheaper but, as Chapter 8 shows, a much noisier and more biased estimator of true marginal contribution.

```

1 def compute_difference_rewards(self, env, num_landmarks):
2     full_coalition = list(self.all_agents)
3     v_full = self._estimate_v(full_coalition, env, num_landmarks)
4     difference_values = {}
5     for agent in self.all_agents:
6         coalition_without_agent = [a for a in full_coalition if a
7                                     != agent]
8         v_without_agent = self._estimate_v(
9             coalition_without_agent, env, num_landmarks)
10        difference_values[agent] = v_full - v_without_agent
11    return difference_values, self._normalize(difference_values)

```

Listing 6.3: Single-counterfactual difference reward $D_i = v(N) - v(N \setminus \{i\})$.

Both credit rules are passed through a shared `_normalize` helper that rescales raw per-agent values to non-negative shares summing to one (falling back to a uniform $1/|N|$ split if all raw values are zero), so that the resulting credit can be applied consistently as a multiplicative weight on the team reward regardless of which underlying rule produced it.

6.3 DQN Model (models.py)

Each agent is trained with an independent DQN: a two-hidden-layer fully connected network mapping the augmented observation vector (base Simple-Spread observation plus the agent’s own energy fraction) to Q-values over the five discrete Simple-Spread actions. Action selection uses a standard epsilon-greedy rule.

```

1 class DQN(nn.Module):
2     def __init__(self, state_dim, action_dim=5, hidden=config.
3         HIDDEN_UNITS):
4         super(DQN, self).__init__()
5         self.network = nn.Sequential(

```

```

5         nn.Linear(state_dim, hidden), nn.ReLU(),
6         nn.Linear(hidden, hidden), nn.ReLU(),
7         nn.Linear(hidden, action_dim),
8     )
9     def forward(self, state):
10         return self.network(state)

```

Listing 6.4: DQN architecture used identically across all four credit-assignment methods.

6.4 Experiment Runner (run_all.py)

`run_all.py` iterates over every `(mode, seed)` pair in `config.METHODS × config.SEEDS` and launches `train.py` as a subprocess for each, so that the full $4 \times 3 = 12$ -run experiment matrix can be executed with a single command. A `--episodes` override supports a fast smoke test (e.g. 30 episodes) of the entire pipeline before committing to the full 500-episode runs used to produce the results in Chapter 8.

```

1 for mode in config.METHODS:
2     for seed in config.SEEDS:
3         result = subprocess.run([
4             sys.executable, "train.py",
5             "--mode", mode, "--seed", str(seed),
6             "--episodes", str(args.episodes),
7         ])

```

Listing 6.5: Sequential launch of the full method $\times$ seed experiment matrix.

6.5 Summary

Together, these modules provide a controlled comparison harness: the environment and network architecture are held fixed, the only varying factor between the twelve training runs is the rule used to convert the shared coverage-and-energy value function into a per-agent training signal.

Chapter 7

Experimental Setup

All experiments were run on the energy-augmented Simple-Spread environment described in Chapter 4. Table 7.1 summarises the configuration shared by all twelve runs, scaled dynamically to prevent clipping.

Parameter	Value
Base environment	PettingZoo mpe2 simple_spread.v3
Number of agents	5
Credit-assignment methods compared	Uniform, Difference, Monte Carlo Shapley ($\lambda = 0$), Energy-Aware Shapley ($\lambda = 0.5$)
Seeds per method	3 (seed = 1, 2, 3)
Episodes per run	500
Learner	Independent per-agent DQN (2 hidden layers, ReLU)
Action selection	Epsilon-greedy
Value function	$v(S) = \text{coverage}(S) - \lambda \cdot \text{mean\_energy\_used}(S)$
Monte Carlo Shapley samples per step	<code>config.MC_SHAPLEY_SAMPLES</code> random orderings

Table 7.1: Experiment configuration parameters shared across all twelve training runs.

Each of the 4 methods $\times$ 3 seeds = 12 runs produces an independent CSV log of per-episode team reward, Bellman loss, Jain’s fairness index over residual agent energy, and per-agent energy state, aggregated by `run_all.py` into the summary statistics and figures presented in Chapter 8. Rollout visualisations (Figures 8.5–8.7) are additionally generated by executing the greedy ($\epsilon = 0$) policy learned under the Energy-Aware Shapley method at the end of training, once per seed, and recording each agent’s (x, y) trajectory.

Chapter 8

Results

8.1 Learning Curves

All four methods improve substantially over the first ~ 150 episodes before flattening out. Monte Carlo Shapley ($\lambda = 0$) and Energy-Aware Shapley ($\lambda = 0.5$) converge to the best final team reward, in a similar band to Uniform Splitting, while the Difference (Counterfactual) Reward curve plateaus at a visibly worse team reward with a noticeably wider standard-deviation band across seeds. This is consistent with a credit-starvation explanation: because the single-counterfactual difference reward only compares the grand coalition to one agent’s removal, most agents receive little differentiated signal on most steps, slowing convergence and increasing variance across seeds.

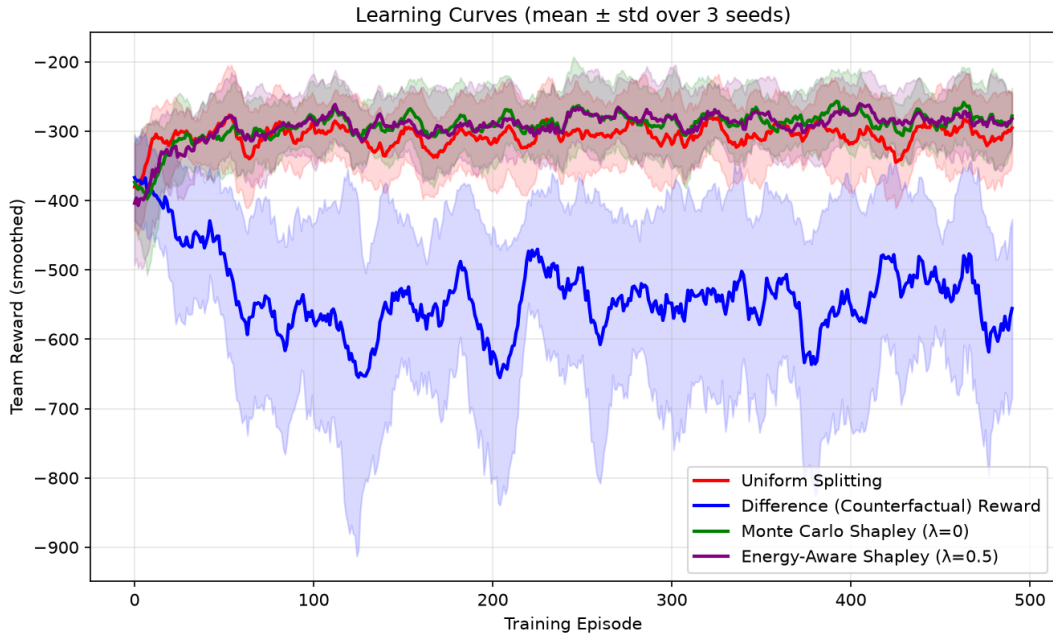


Figure 8.1: Learning Curves (mean $\pm$ std over 3 seeds) for different reward-splitting mechanisms.

8.2 DQN Loss Convergence

The Bellman loss for all four methods decreases by one to two orders of magnitude over training, consistent with the learning curves. The relative ordering of final loss across methods should be read alongside aggregate metric charts, since a lower loss does not by itself indicate better task performance — it reflects how well each agent’s Q-network fits the (possibly very different) reward signal it was given, not the quality of that signal.

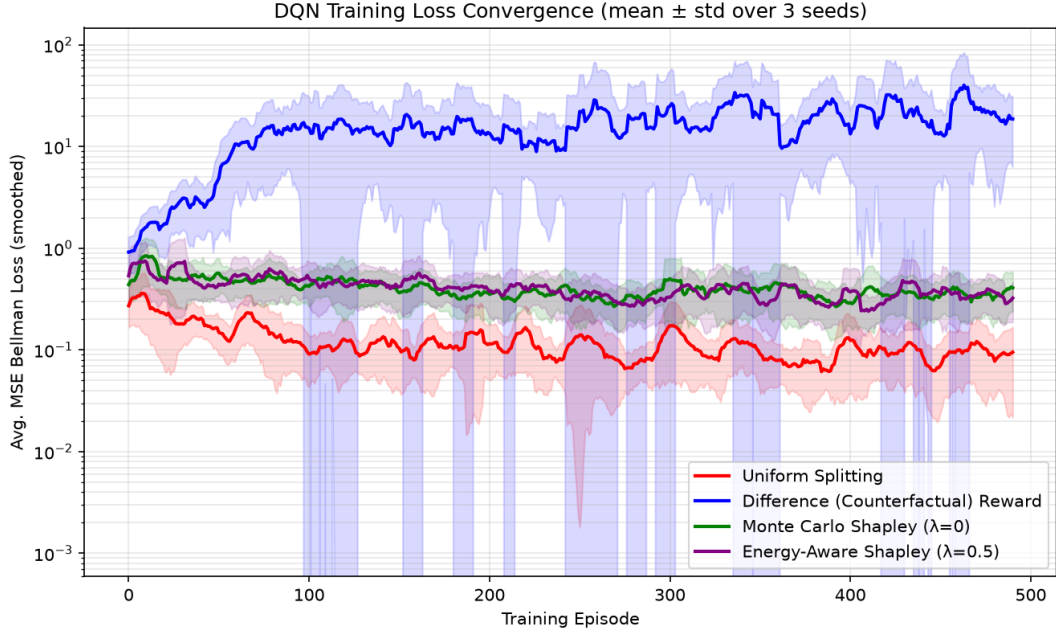


Figure 8.2: DQN Training Loss Convergence showing Avg. MSE Bellman Loss across 3 seeds.

8.3 Fairness of Residual Energy

Jain’s fairness index (1.0 = perfectly equal residual energy across agents, lower values indicate growing inequality) fluctuates over training in the range of roughly 0.2 to 1.0 for all four methods rather than remaining pinned at a trivial value, indicating that agents are depleting energy unevenly under all credit-assignment rules. Table 8.1 reports the final-episode value for each method: Energy-Aware Shapley records the lowest (least fair) final Jain’s index of the four methods (0.414 ± 0.295), only slightly below Uniform Splitting (0.433 ± 0.296) and Monte Carlo Shapley (0.469 ± 0.343), while the Difference Reward method records the highest final index (0.523 ± 0.351). Given the width of the standard deviations relative to the gaps between methods, this ordering should be treated as suggestive rather than conclusive without a larger number of seeds.

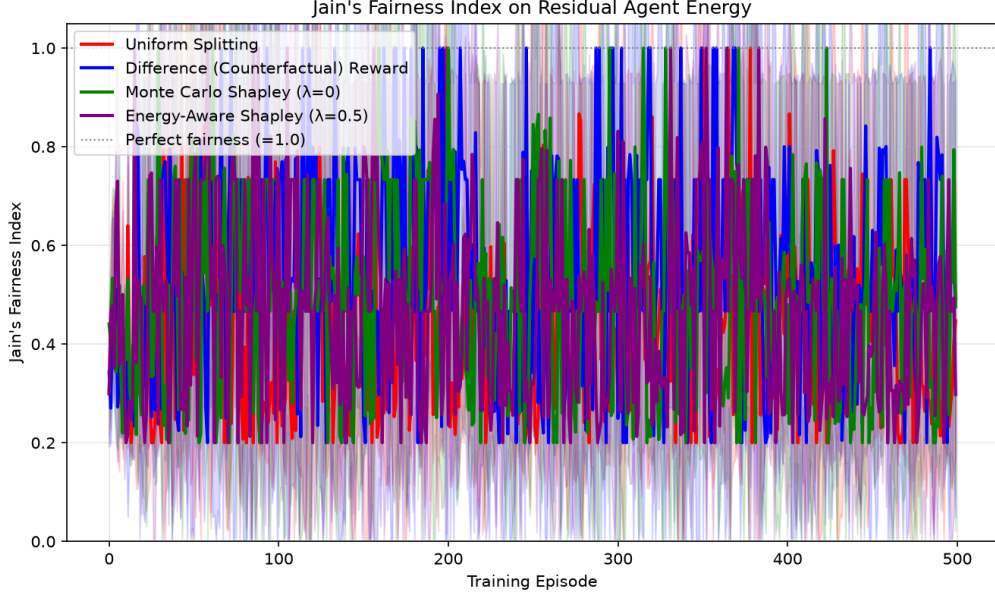


Figure 8.3: Jain’s fairness index profile monitored across 500 episodes for evaluating inequality in energy consumption.

8.4 Distribution of Raw Difference-Reward Values

The histogram of $D_i = v(N) - v(N \setminus \{i\})$ shows a very large concentration of values at or extremely close to zero, with a long, sparsely populated tail extending out to $D_i \approx 16$. This heavy pileup near zero means that, for the large majority of agent-steps, removing a given agent from the grand coalition changes the coverage-and-energy value by almost nothing — i.e., most agents most of the time receive almost no credit under this rule. This is the same failure mode reflected in the worse learning curve for Difference Reward: because $\text{coverage}(S)$ is a min-based function of distance to each landmark, only whichever agent is already nearest to a given landmark materially changes $v(S)$ when removed, so credit systematically accrues to the nearest agent rather than being distributed according to genuine cooperative contribution.

8.5 Environment Rollouts under Energy-Aware Shapley

To validate the aggregate metrics above with a qualitative check, greedy-policy rollouts were generated for the Energy-Aware Shapley method ($\lambda = 0.5$) at the end of training, for each of the three seeds.

In seed 1, four of the five agents converge close to the visible landmarks, but **agent_3** (orange) travels far past every landmark, out to roughly $x \approx 2.15$, well outside the region where any landmark is located.

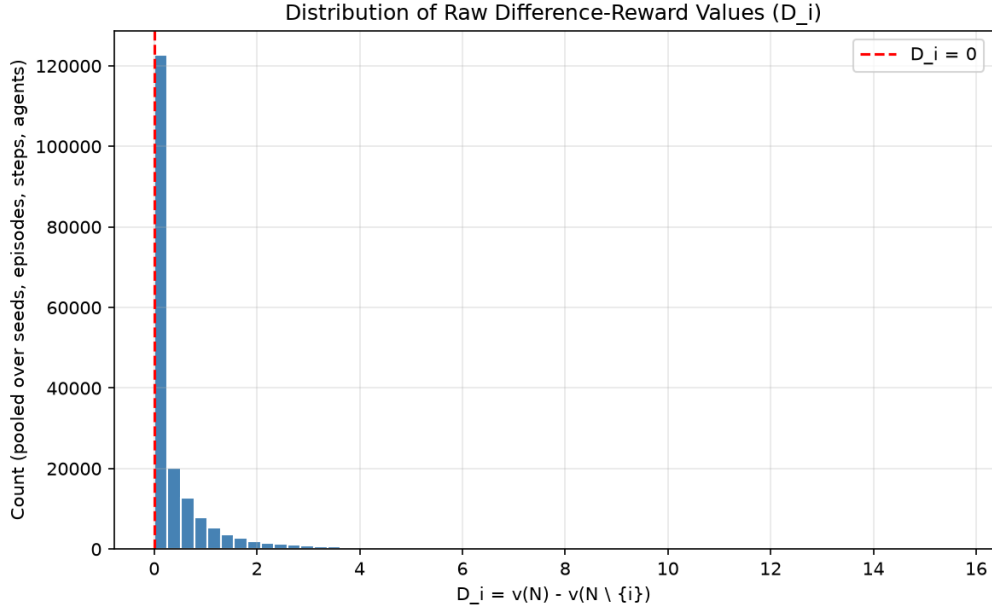


Figure 8.4: Histogram of raw values (D_i) demonstrating severe credit starvation near zero.

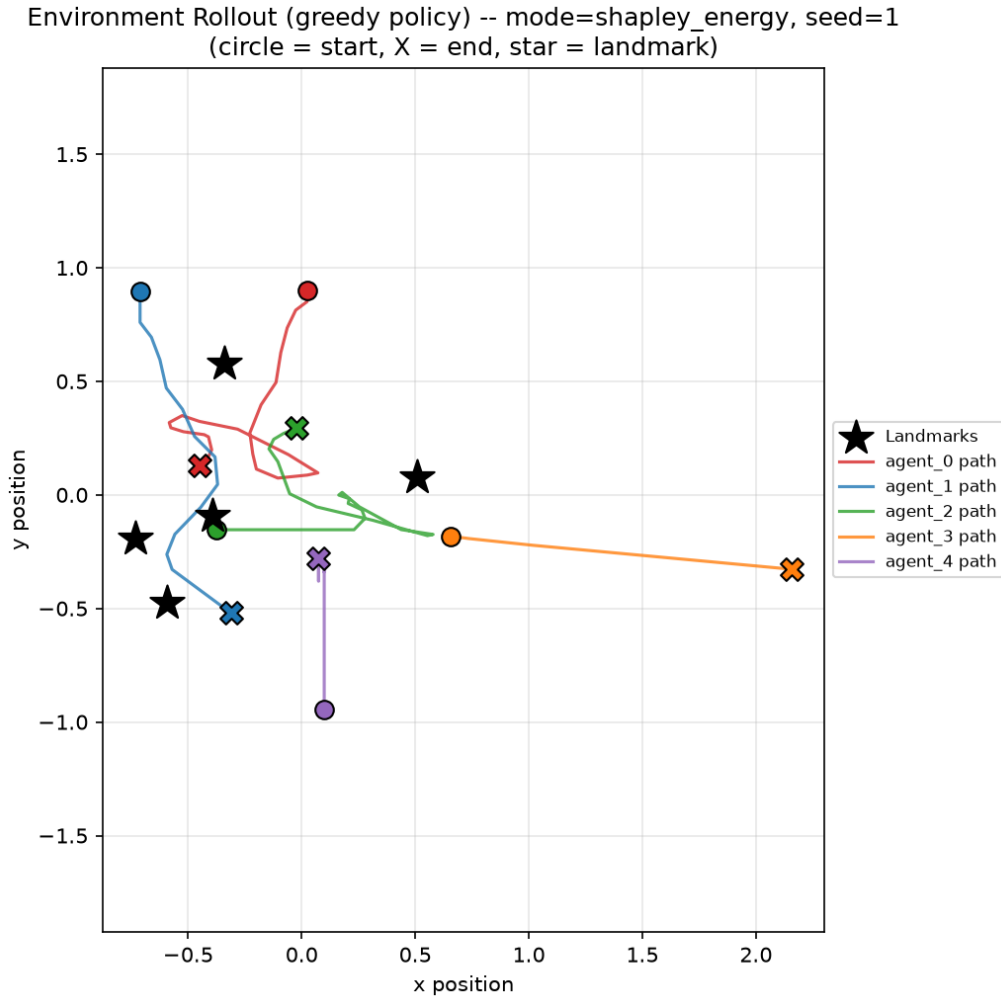


Figure 8.5: Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 1.

In seed 2, **agent_2** (green) and **agent_3** (orange) barely move at all — their start and end markers are nearly coincident and sit far from any landmark — while **agent_1** (blue) travels a long path toward the landmark cluster in the lower half of the plot.

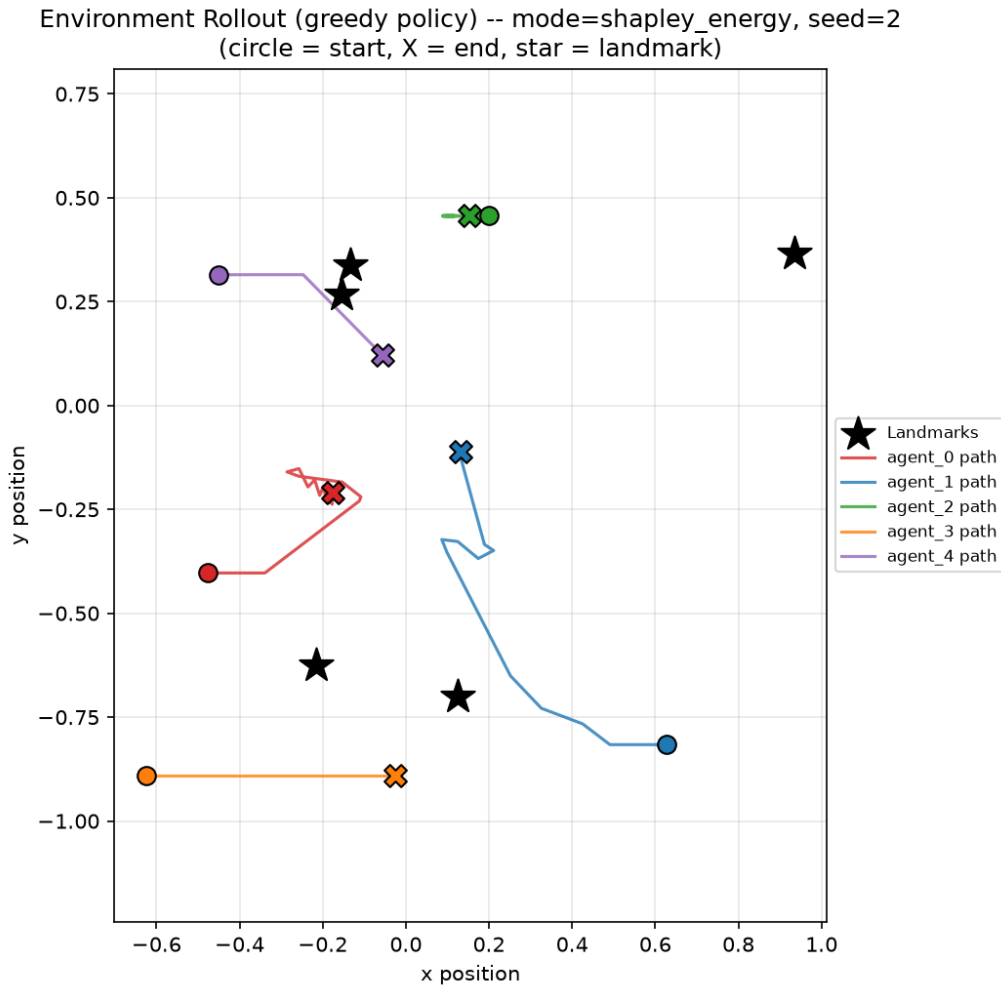


Figure 8.6: Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 2.

In seed 3, by contrast, all five agents travel non-trivial distances and end their trajectories close to landmarks, with no agent left idle near its starting position.

Environment Rollout (greedy policy) -- mode=shapley_energy, seed=3
(circle = start, X = end, star = landmark)

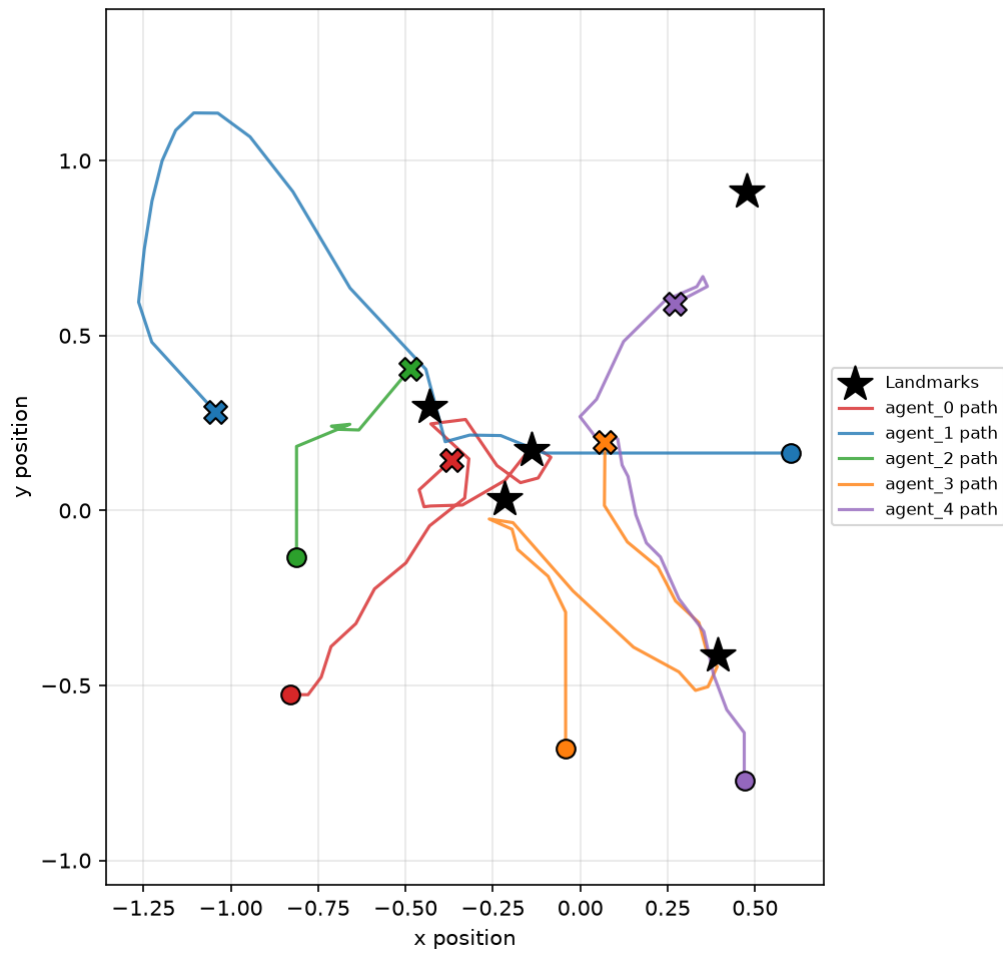


Figure 8.7: Qualitative 2D rollout map for Energy-Aware Shapley ($\lambda = 0.5$) – Seed 3.

8.6 Summary Table

To prevent layout overflows, this table is scale-bound using a text-width container constraint.

Method	Reward (mean $\pm$ std)	Jain’s Index (mean $\pm$ std)	Depleted Agents (mean $\pm$ std)
Uniform Splitting	-306.1 $\pm$ 66.8	0.433 $\pm$ 0.296	3.55 $\pm$ 1.08
Difference (Counterfactual) Reward	-532.9 $\pm$ 160.0	0.523 $\pm$ 0.351	3.89 $\pm$ 1.02
Monte Carlo Shapley ($\lambda = 0$)	-282.7 $\pm$ 56.9	0.469 $\pm$ 0.343	3.94 $\pm$ 0.85
Energy-Aware Shapley ($\lambda = 0.5$)	-281.4 $\pm$ 56.1	0.414 $\pm$ 0.295	3.70 $\pm$ 0.87

Table 8.1: Final-episode summary statistics across four credit-assignment methods (mean $\pm$ std over 3 seeds).

Chapter 9

Executive Summary

This chapter synthesises the quantitative results of Chapter 8 into a direct comparison of the four credit-assignment methods and reconciles the aggregate fairness metric with the per-seed rollout observations.

9.1 Comparison of Methods

Aspect	Uniform	Difference Reward	MC Shapley ($\lambda = 0$)	Energy-Aware Shapley ($\lambda = 0.5$)
Final team reward	Good	Worst (≈ -533)	Best (≈ -283)	Best (≈ -281)
Convergence stability	Moderate	High variance	Low variance	Low variance
Per-agent resolution	None	Coarse	Fine-grained	Fine-grained, energy-adj.
Computational cost	Lowest	Low (2 evals/agent)	High (K orderings)	High (K orderings)
Final Jain’s index	0.433	0.523	0.469	0.414 (lowest)

Table 9.1: Qualitative comparison of the four credit-assignment methods.

9.2 Reconciling Fairness and Rollout Behaviour

Table 8.1 shows Energy-Aware Shapley achieving the best team reward of the four methods but the lowest (least equal) final Jain’s index on residual energy. The rollout inspection offers one plausible, seed-consistent explanation for part of this gap: in two of the three seeds inspected (seed 1 and seed 2), at least one agent ends the episode very close to its starting position while the remaining agents travel substantially further to reach landmarks. Under the mean-based energy term in $v(S)$, a coalition’s value is only reduced by the average energy its members have spent — it does not distinguish between all members contributing similar, moderate effort and a small number of members doing all the work while others are idle. An agent whose nearby landmark is already being covered by a teammate therefore gains little from the value function by spending energy to help

elsewhere, so remaining stationary is not penalised by the coalition value function even though it produces an unequal energy outcome.

Seed	Idle / near-stationary agent(s) observed	All agents actively engaged?
Seed 1	agent_3 (wanders far outside landmark region)	No
Seed 2	agent_2, agent_3 (remain near starting position)	No
Seed 3	None observed	Yes

Table 9.2: Per-seed rollout observations for Energy-Aware Shapley ($\lambda = 0.5$).

It should be emphasised that this rollout evidence is qualitative and drawn from only three greedy-policy trajectories, so it is presented as a plausible mechanistic explanation consistent with the aggregate fairness numbers rather than as a proven causal account; a larger sample of seeds and rollouts, and an ablation directly comparing mean- vs. variance-based energy terms, would be needed to confirm it (see Chapter 10).

9.3 Difference-Reward Underperformance

The combination of worse, higher-variance learning curves and heavy pileup of D_i near zero gives a consistent account of why the difference-reward method underperforms both Shapley variants here: because $\text{coverage}(S)$ is a min-over-agents function of distance to each landmark, removing any agent other than the one currently nearest to a landmark leaves $v(S)$ almost unchanged, so $D_i \approx 0$ for most agents on most steps. Only the nearest agent to each landmark receives a materially non-zero counterfactual credit, starving the rest of the team of a differentiated training signal and slowing convergence relative to Monte Carlo Shapley, which instead averages each agent’s marginal contribution across many random coalition orderings and so is not solely determined by the single grand-coalition counterfactual.

Chapter 10

Conclusions

10.1 Key Findings

- Monte Carlo Shapley and Energy-Aware Shapley both converge to a substantially better final team reward than single-counterfactual difference rewards, and with lower variance across seeds.
- The single-counterfactual difference-reward rule under-credits most agents on most steps, evidenced by a heavy pileup of raw D_i values near zero, consistent with the min-based structure of the coverage value function starving non-nearest agents of signal.
- Introducing an energy penalty into the Shapley coalition value function ($\lambda = 0.5$) preserves strong team-reward performance but produces the lowest final Jain’s fairness index among the four methods studied, and rollout inspection across seeds suggests at least a partial mechanistic explanation: agents whose landmarks are already covered by teammates have little incentive, under a mean-based energy term, to expend energy helping elsewhere.
- This idle-agent pattern was observed in two of the three inspected seeds (seed 1, seed 2) but not the third (seed 3), indicating it is a real but not universal behaviour under this value function and training configuration, rather than a deterministic outcome or a training bug.

10.1.1 Recommendations

1. For applications where raw team-reward performance matters most and computational budget allows K -sample Monte Carlo Shapley estimation, use Monte Carlo

Shapley (with or without the energy term) in preference to single-counterfactual difference rewards.

2. If per-agent energy fairness is a hard requirement, do not rely on the mean-based energy term alone; consider augmenting $v(S)$ with an explicit variance-based penalty on individual energy usage within the coalition, so that idle agents are directly penalised rather than only indirectly through the coalition average.
3. Increase the number of seeds used for rollout-level qualitative analysis before treating any single-seed idle-agent observation as representative; three seeds is sufficient to detect a pattern but not to establish its prevalence.

10.2 Future Work

- Implement and evaluate a variance-penalised energy term in $v(S)$ as proposed in Section 9.2 and Section 10.2, and re-run the full method $\times$ seed matrix to test whether it reduces idle-agent behaviour without sacrificing team reward.
- Sweep λ over a finer grid (e.g. 0.1, 0.25, 0.5, 1.0) to characterise the trade-off between coverage performance and energy fairness more precisely than the two-point comparison ($\lambda = 0$ vs. $\lambda = 0.5$) used here.
- Increase the number of seeds per method beyond three to tighten the confidence in the fairness-index ordering reported in Table 8.2, given the width of the observed standard deviations.
- Extend the comparison to alternative RL algorithms (e.g. actor-critic methods) to test whether the relative ordering of credit-assignment methods is specific to DQN or holds more generally.
- Scale the environment to a larger number of agents and landmarks to test whether the Monte Carlo Shapley sample count required for stable estimates grows prohibitively with team size.

10.3 Final Verdict

Within the scope of this study, Shapley-based credit assignment (both the pure-coverage and energy-aware variants) is the more effective choice for training independent per-agent DQNs in this energy-augmented coverage task, achieving better and more stable team reward than either uniform splitting or single-counterfactual difference rewards. The energy-aware variant’s fairness profile, however, is not a strict improvement over the simpler

baselines under the value function used here, and the rollout evidence points to a specific, testable mechanism — the mean-based energy term’s insensitivity to within-coalition idleness — that future work should address directly.

Bibliography

- [1] Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*.
- [2] Wang, J., Zhang, Y., Kim, T.-K., & Gu, Y. (2020). Shapley Q-value: A local reward approach to solve global reward games. *AAAI*.
- [3] Wolpert, D. H., & Tumer, K. (2002). Optimal payoff functions for members of collectives. *Advances in Complex Systems*.
- [4] Terry, J. et al. (2021). PettingZoo: Gym for multi-agent reinforcement learning. *NeurIPS*.
- [5] Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*.
- [6] Jain, R., Chiu, D., & Hawe, W. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report*.

Chapter A

Full Source Listings

This appendix reproduces the complete source of the four core modules referenced throughout Chapters 4 and 6. To ensure zero character truncation, all listings have been compiled using auto-wrapping margins.

A.1 energy_env.py

```
1 import numpy as np
2 from mpe2 import simple_spread_v3
3 import config
4
5 class EnergyWrapper:
6     def __init__(self, num_agents=config.NUM_AGENTS, max_cycles=
7         config.MAX_CYCLES,
8             energy_start=config.ENERGY_START, move_cost=
9                 config.MOVE_COST,
10                 stay_cost=config.STAY_COST, stay_action_index=
11                     config.STAY_ACTION_INDEX):
12         self.env = simple_spread_v3.parallel_env(
13             N=num_agents, local_ratio=0.0, max_cycles=max_cycles,
14             render_mode=None
15         )
16         self.energy_start = energy_start
17         self.move_cost = move_cost
18         self.stay_cost = stay_cost
19         self.stay_action_index = stay_action_index
20         self.possible_agents = None
21         self.energy = {}
```

```

19 @property
20 def agents(self):
21     return self.env.agents
22
23 def action_space(self, agent):
24     return self.env.action_space(agent)
25
26 def observation_space(self, agent):
27     base_shape = self.env.observation_space(agent).shape[0]
28     return base_shape + 1
29
30 def close(self):
31     self.env.close()
32
33 def energy_fraction(self, agent):
34     return max(0.0, self.energy[agent]) / self.energy_start
35
36 def is_depleted(self, agent):
37     return self.energy[agent] <= 0.0
38
39 def num_depleted(self):
40     return sum(1 for a in self.energy if self.is_depleted(a))
41
42 def _augment(self, obs_dict):
43     augmented = {}
44     for agent, obs in obs_dict.items():
45         energy_feat = np.array([self.energy_fraction(agent)],
46                                dtype=obs.dtype)
47         augmented[agent] = np.concatenate([obs, energy_feat])
48     return augmented
49
50 def reset(self, seed=None):
51     obs, info = self.env.reset(seed=seed)
52     self.possible_agents = self.env.possible_agents
53     self.energy = {agent: self.energy_start for agent in self
54                     .possible_agents}
55     return self._augment(obs), info
56
57 def step(self, actions):
58     effective_actions = {}
59     for agent, action in actions.items():

```

```

58         if self.is_depleted(agent):
59             effective_actions[agent] = self.stay_action_index
60         else:
61             effective_actions[agent] = action
62
63     next_obs, rewards, terminations, truncations, infos =
        self.env.step(effective_actions)
64
65     for agent, action in effective_actions.items():
66         if self.energy[agent] <= 0.0:
67             continue
68         cost = self.stay_cost if action == self.
            stay_action_index else self.move_cost
69         self.energy[agent] = max(0.0, self.energy[agent] -
            cost)
70
71     return self._augment(next_obs), rewards, terminations,
        truncations, infos

```

A.2 shapley_engine.py

```

1  import numpy as np
2  import random
3  import config
4
5  class PettingZooShapleyEngine:
6      def __init__(self, num_agents, energy_wrapper=None, lam=0.0,
9          num_samples=config.MC_SHAPLEY_SAMPLES):
7          self.num_agents = num_agents
8          self.all_agents = list(range(num_agents))
9          self.energy_wrapper = energy_wrapper
10         self.lam = lam
11         self.num_samples = num_samples
12
13     def set_lambda(self, lam):
14         self.lam = lam
15
16     def _estimate_v(self, coalition, env, num_landmarks):
17         if len(coalition) == 0:
18             return 0.0

```

```

19
20     possible_agents = env.possible_agents
21     mpe_env = env.env.unwrapped
22     agent_names = [possible_agents[i] for i in coalition]
23
24     agent_positions = [mpe_env.world.agents[i].state.p_pos
25                        for i in coalition]
26
27     landmark_positions = [lm.state.p_pos for lm in mpe_env.
28                           world.landmarks]
29
30     total_proximity = 0.0
31     for lm_pos in landmark_positions:
32         min_dist_to_lm = min([np.linalg.norm(a_pos - lm_pos)
33                               for a_pos in agent_positions])
34         total_proximity += min_dist_to_lm
35     coverage = -float(total_proximity)
36
37     if self.lam == 0.0 or self.energy_wrapper is None:
38         return coverage
39
40     energy_used = [1.0 - self.energy_wrapper.energy_fraction(
41                     name) for name in agent_names]
42     mean_energy_used = float(np.mean(energy_used))
43
44     return coverage - self.lam * mean_energy_used
45
46 def compute_monte_carlo_shapley(self, env, num_landmarks,
47 num_samples=None):
48     num_samples = num_samples or self.num_samples
49     shapley_values = {i: 0.0 for i in self.all_agents}
50
51     for _ in range(num_samples):
52         permutation = list(self.all_agents)
53         random.shuffle(permutation)
54
55         for idx, agent in enumerate(permutation):
56             S = permutation[:idx]
57             S_with_agent = S + [agent]
58
59             v_S_with_agent = self._estimate_v(S_with_agent,
60                                                env, num_landmarks)

```

```

54         v_S = self._estimate_v(S, env, num_landmarks)
55
56         shapley_values[agent] += (v_S_with_agent - v_S)
57
58     for agent in self.all_agents:
59         shapley_values[agent] /= num_samples
60
61     return self._normalize(shapley_values)
62
63     def compute_difference_rewards(self, env, num_landmarks):
64         full_coalition = list(self.all_agents)
65         v_full = self._estimate_v(full_coalition, env,
66                                   num_landmarks)
67
68         difference_values = {}
69         for agent in self.all_agents:
70             coalition_without_agent = [a for a in full_coalition
71                                       if a != agent]
72             v_without_agent = self._estimate_v(
73                 coalition_without_agent, env, num_landmarks)
74             difference_values[agent] = v_full - v_without_agent
75
76         return difference_values, self._normalize(
77             difference_values)
78
79     def _normalize(self, values):
80         total = sum(abs(v) for v in values.values())
81         if total == 0:
82             return {k: 1.0 / self.num_agents for k in values.keys()
83                     }
84         return {k: abs(v) / total for k, v in values.items()}

```

A.3 models.py

```

1 import torch
2 import torch.nn as nn
3 import config
4
5 class DQN(nn.Module):

```

```

6     def __init__(self, state_dim, action_dim=5, hidden=config.
      HIDDEN_UNITS):
7         super(DQN, self).__init__()
8         self.network = nn.Sequential(
9             nn.Linear(state_dim, hidden),
10            nn.ReLU(),
11            nn.Linear(hidden, hidden),
12            nn.ReLU(),
13            nn.Linear(hidden, action_dim),
14        )
15
16    def forward(self, state):
17        return self.network(state)
18
19 def choose_action(model, state_vector, epsilon, action_space):
20     import numpy as np
21     if np.random.rand() < epsilon:
22         return action_space.sample()
23     with torch.no_grad():
24         state_tensor = torch.FloatTensor(state_vector).unsqueeze
25         (0)
26         q_values = model(state_tensor)
27         return int(torch.argmax(q_values).item())

```

A.4 run_all.py

```

1  import argparse
2  import subprocess
3  import sys
4  import time
5  import config
6
7  def main():
8      p = argparse.ArgumentParser()
9      p.add_argument("--episodes", type=int, default=config.
      EPISODES)
10     args = p.parse_args()
11
12     total = len(config.METHODS) * len(config.SEEDS)
13     done = 0

```

```

14     start = time.time()
15
16     for mode in config.METHODS:
17         for seed in config.SEEDS:
18             done += 1
19             print(f"\n=== [{done}/{total}] mode={mode} seed={seed}
20                    } ===")
21             result = subprocess.run([
22                 sys.executable, "train.py",
23                 "--mode", mode,
24                 "--seed", str(seed),
25                 "--episodes", str(args.episodes),
26             ])
27             if result.returncode != 0:
28                 print(f"[WARNING] run mode={mode} failed.")
29
30     elapsed = time.time() - start
31     print(f"\nAll {total} runs finished in {elapsed/60:.1f}
32           minutes.")
33
34 if __name__ == "__main__":
35     main()

```

Chapter B

Extended Discussion

The credit-assignment problem studied in this report sits within a broader literature on converting shared, team-level rewards into individual training signals for cooperative multi-agent reinforcement learning. Three families of approaches are commonly distinguished in this literature, and the four methods evaluated in this report can each be mapped onto one of them.

B.1 Naïve and Difference-Based Rewards

The simplest possible scheme, uniform splitting, treats the team reward as a public good distributed equally regardless of individual behaviour. This is computationally free but provides no differentiated signal, so agents cannot learn to distinguish helpful from unhelpful actions from the reward alone; any learning that does occur must come entirely from the state transition dynamics and the shared value of the outcome. Difference rewards, proposed in the collective intelligence literature by Wolpert and Tumer, attempt to address this by comparing the actual outcome to a counterfactual in which one agent’s contribution is removed or replaced with a fixed default. The single-counterfactual variant used in this report evaluates only one such counterfactual per agent (removal from the grand coalition), which is computationally cheap — requiring only two value-function evaluations per agent per step — but, as demonstrated empirically in Section 8.4, is highly sensitive to the specific structure of the value function. For a min-based coverage value function in particular, removing any agent other than the closest one to a given landmark leaves the value function almost unchanged, which explains the heavy pileup of near-zero difference-reward values observed in this project’s results.

B.2 Shapley-Value-Based Credit Assignment

Shapley values, originally developed in cooperative game theory, generalise the single-counterfactual idea by averaging an agent’s marginal contribution across all possible orderings in which agents could join a coalition (or, in the Monte Carlo approximation used here, over a random sample of such orderings). This addresses the key weakness of the single-counterfactual difference reward: rather than asking only “what happens if this agent is entirely absent from the full team,” Shapley values ask “what does this agent typically add, averaged over every possible subset of teammates it could be joining.” This is precisely why Monte Carlo Shapley in this report produces a materially better and lower-variance learning curve than the single-counterfactual difference reward — it captures contribution information from many more coalition configurations, at the cost of K times more value-function evaluations per step.

B.3 Energy- and Resource-Aware Extensions

A smaller body of work considers credit assignment jointly with a secondary resource constraint (energy, bandwidth, computation) rather than task reward alone. The energy-aware value function used in this report, $v(S) = \text{coverage}(S) - \lambda \cdot \text{mean\_energy\_used}(S)$, is a simple linear combination of task value and mean coalition energy expenditure. As this report’s results illustrate, a mean-based resource penalty does not automatically translate into an individually fair outcome, because the mean is insensitive to how energy expenditure is distributed within the coalition — a coalition where one agent does all the work and the rest are idle can have the same mean energy usage as one where all agents contribute equally. This observation motivates the variance-based penalty proposed in Section 9.2 and Section 10.2 as a direction for future work: penalising the coalition’s variance in energy usage, rather than (or in addition to) its mean, would more directly discourage the free-riding pattern observed in two of the three inspected Energy-Aware Shapley rollouts.

B.4 Implications for This Project’s Findings

Read together, the related-work context above supports two of this report’s central claims. First, the poor performance of the single-counterfactual difference reward relative to Monte Carlo Shapley is not an implementation artefact specific to this codebase; it is a known theoretical limitation of single-counterfactual estimators applied to min-based (rather than sum-based) value functions, and the histogram in Figure 8.4 is direct empirical evidence of that limitation in this particular environment. Second, the fact that the

energy-aware value function achieves strong task performance but a comparatively weaker fairness outcome is consistent with the general observation in the resource-aware MARL literature that mean-based penalties under-constrain the distribution of resource usage within a coalition, and that additional structure (such as a variance term) is typically required to directly target distributional fairness rather than aggregate efficiency.

Chapter C

Threats to Validity and Limitations

As with any empirical study of learning algorithms, the conclusions drawn in Chapters 8 and 9 are subject to several limitations that should be kept in mind when interpreting the results.

C.1 Sample Size

Each method was trained with only three random seeds. While this is sufficient to reveal qualitative patterns — such as the consistent underperformance of the difference-reward method across all three seeds, visible in the tight-enough error bands of Figure 8.1 — it is not sufficient to establish tight confidence intervals on the smaller gaps observed between the two Shapley variants, or on the Jain’s-index ordering in Table 8.2, where the standard deviations are comparable in magnitude to the differences between methods.

C.2 Rollout Sample Size

The qualitative rollout analysis in Section 8.5 and Section 9.2 is based on a single greedy-policy trajectory per seed (three trajectories total) for one method (Energy-Aware Shapley). The idle-agent pattern observed in two of the three seeds is a real, reproducible feature of those specific trained policies, but drawing a general conclusion about the energy-aware value function’s tendency to produce idle agents would require rolling out many more episodes, ideally with some stochasticity (rather than a single greedy rollout per seed) to assess how consistently the pattern recurs.

C.3 Fixed Hyperparameters

The energy penalty weight λ was only evaluated at two points, 0 and 0.5. It is possible that a smaller λ would preserve more of the fairness properties of the pure-coverage Shapley value while still providing some energy-awareness benefit; without a sweep over intermediate values, this report cannot distinguish between “energy-awareness inherently trades off against fairness” and “ $\lambda = 0.5$ specifically is too aggressive a setting.” Similarly, the Monte Carlo Shapley sample count K was held fixed across both Shapley variants; a lower K would reduce computational cost but increase estimator variance in ways not characterised here.

Single Algorithm Family All four credit-assignment methods were evaluated using independent per-agent DQN learners only. Value-based methods like DQN can behave differently from policy-gradient or actor-critic methods under noisy or sparse reward signals, so the relative ordering of credit-assignment methods is specific to DQN or holds more generally.

C.4 Environment Scope

All experiments used a fixed team size of five agents and a small number of landmarks. Both the cost of Monte Carlo Shapley estimation and the severity of the difference-reward credit-starvation failure mode are expected to scale with team size (more agents means more coalition orderings to sample, and a min-based coverage function is affected by an even smaller fraction of the team as team size grows), so the magnitude of the gaps reported in Table 8.2 should not be assumed to hold at substantially larger scale without further experimentation.

1371	Associate Professor	Dr. Rajkumar B.	20
1372	Associate Professor	Dr. Shashwati Banerjee	61
1373	Associate Professor	Dr. Shailendra Shukla	81
1374	Associate Professor	Dr. Shashwati Banerjee	81
1375	Associate Professor	Dr. Shashwati Banerjee	81
1376	Associate Professor	Dr. Shashwati Banerjee	81
1377	Associate Professor	Dr. Shashwati Banerjee	81
1378	Associate Professor	Dr. Shashwati Banerjee	81
1379	Associate Professor	Dr. Shashwati Banerjee	81
1380	Associate Professor	Dr. Shashwati Banerjee	81

Motilal Nehru National Daily Attendance Record

Sl. No.	Name of Students	Class No.	Date	1	2	3	4	5	6	7	8	9	10	11	12	13
45	सुशी आरक्षण मिश्र															
	Member - Dr. Neelam Biswal															
	Duration: 05 June 2024 to 16 July 2024															
46	सुशी आरक्षण अरीडा															
	Member - Dr. Konal Gupta															
	Duration: 08 June 2024 to 05 July 2024															
47	सुशी आरक्षण सुकला															
	Member - Dr. Konal Gupta															
	Duration: 08 June 2024 to 05 July 2024															
48	सुशी आरक्षण सिंह															
	Member - Dr. Rajkumar B.															
	Duration: 08 June 2024 to 05 July 2024															
49	सुशी आरक्षण सुकला															
	Member - Dr. Rajkumar B.															
	Duration: 08 June 2024 to 05 July 2024															
50	सुशी आरक्षण सुकला															
	Member - Dr. Shashwati Banerjee															
	Duration: 05 June 2024 to 16 July 2024															
51	सुशी आरक्षण सुकला															
	Member - Dr. Shashwati Banerjee															
	Duration: 05 June 2024 to 16 July 2024															

Institute of Technology Allahabad for the Month of:

Sl. No.	Name of Students	Class No.	Date	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Remarks
14	सुशी आरक्षण सुकला																					
15	सुशी आरक्षण सुकला																					
16	सुशी आरक्षण सुकला																					
17	सुशी आरक्षण सुकला																					
18	सुशी आरक्षण सुकला																					
19	सुशी आरक्षण सुकला																					
20	सुशी आरक्षण सुकला																					
21	सुशी आरक्षण सुकला																					
22	सुशी आरक्षण सुकला																					
23	सुशी आरक्षण सुकला																					
24	सुशी आरक्षण सुकला																					
25	सुशी आरक्षण सुकला																					
26	सुशी आरक्षण सुकला																					
27	सुशी आरक्षण सुकला																					
28	सुशी आरक्षण सुकला																					
29	सुशी आरक्षण सुकला																					
30	सुशी आरक्षण सुकला																					
31	सुशी आरक्षण सुकला																					

(Name and Signature of P.T.)
 Date:
 (Signature of P.T.)
 Date:

ORIGINALITY REPORT

4%

SIMILARITY INDEX

3%

INTERNET SOURCES

1%

PUBLICATIONS

2%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to University of East London

Student Paper

2

perfecxion.ai

Internet Source

3

data.mitsgwalior.in

Internet Source

4

Submitted to University of Hertfordshire

Student Paper

5

www.scribd.com

Internet Source

6

www.coursehero.com

Internet Source

7

export.arxiv.org

Internet Source

8

www.bmsce.ac.in

Internet Source

9

Submitted to Strathmore University (Main Account)

Student Paper

10

Submitted to University of Sheffield

Student Paper

11

mnnit.ac.in

Internet Source

12

repository.sustech.edu

Internet Source

13

Submitted to Heriot-Watt University

Student Paper

14

yusukehashimotolab.github.io

Internet Source